

A decorative graphic on the left side of the slide consisting of two overlapping parallelograms. The front one is blue and the back one is a light greenish-blue. They are positioned diagonally, with the blue one partially covering the green one.

453rd EWS Neural Network for System Identification

Caleb Vourazeris



Background

Inside Work

- Salisbury University Graduate | 4.0 GPA
 - SMART Scholar
 - Physics Bachelors
 - Summa Cum Laude
 - Excellence in engineering award
- University of Maryland Graduate | 3.864 GPA
 - Electrical Engineering Bachelors
- Intern at the 57th Intelligence Squadron
 - Combat Support Database Analyst
- Full time engineer at the 453rd Electronic Warfare Squadron
 - Initially IMOM (improved many on many) team
 - Lacked guidance, tasking, urgency, and engineering
 - Self taught software engineering
 - MATLAB courses, transitioned into python
 - Created a vision for AI/ML development at 453 EWS
 - Began building model for system identification

Outside Work

- Brazilian Jiu Jitsu
 - White Belt (Blue Belt Test 3/30/2025)
 - Consistently trained for 1 year +
 - 3rd Place at Houston Fall International Open IBJJF Jiu Jitsu Championship
 - 3rd Place at Dallas Winter International Open IBJJF Jiu Jitsu Championship 2025
- Guitar
 - Gig performance
 - Weekly lessons
- Poker
 - Playing since 2022
 - Play regularly for fun
 - Low stakes live poker
- Investing
 - Personally managed portfolio
 - Investing since 2019
- Weight Lifting
 - Lift regularly to stay in shape and socialize



Project Scope and Goal

- Inception
 - Drew connection between MATLAB course datasets and 453EWS datasets
 - Analysts work was simply a classification problem
 - Decided to attempt to create a solution
- Problem
 - Automatically characterize signals based on parameters
 - Current Process
 - Rule based flagging system
 - Flags signals were manually inspected
- Issues with current process
 - Not working, a large majority of data not being processed
 - Rules based system is not robust
 - Manual inspection is not reliable (using geo data)
- Impact
 - Process all data that came through the pipeline
 - Automatically characterize signals
 - Robust solution

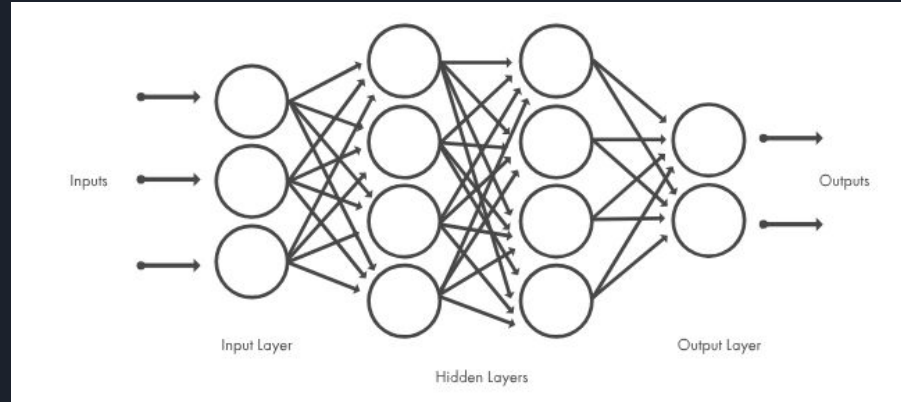


Responsibility

- Project Design Theory
 - Neural Networks
 - Classify handwritten numbers
 - 3 Blue 1 Brown Video Series: Neural Networks
- Project Design Implementation
 - Mock Code Written Using Iris Dataset
 - Code written using Pytorch, Numpy, Pandas etc
- Neural Network for System Identification
 - Project Summary
 - Results

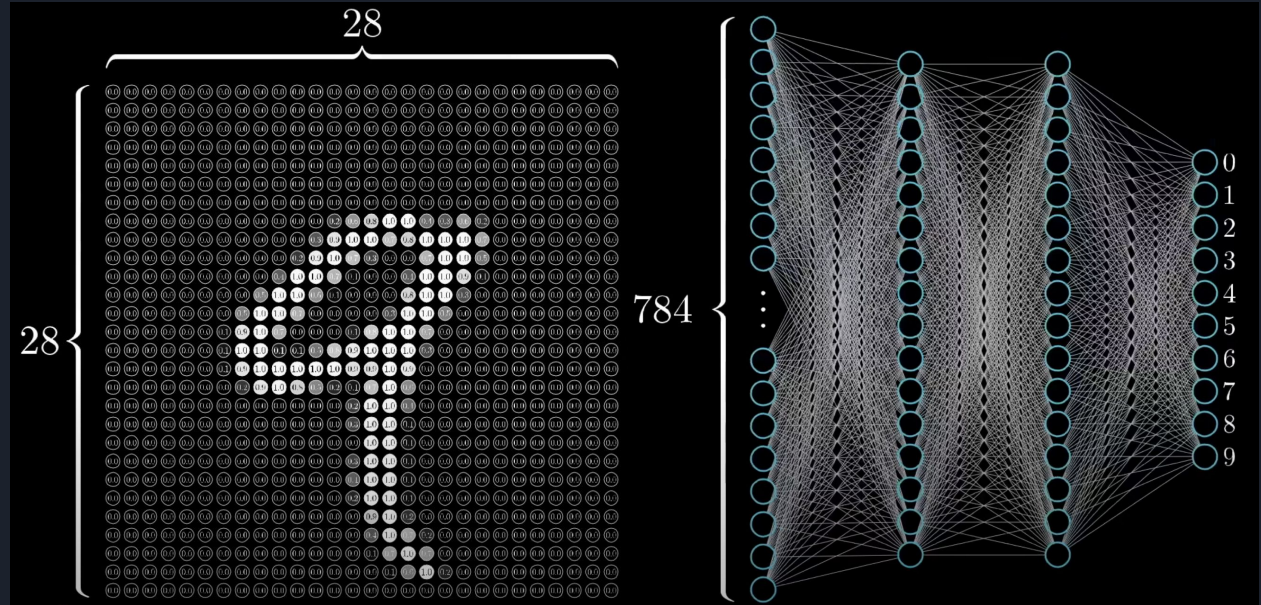
Project Design Theory

- Network Structure
- Neuron Weights
- Network Parameters
- Cost Function
- Gradient Descent
- Backpropagation



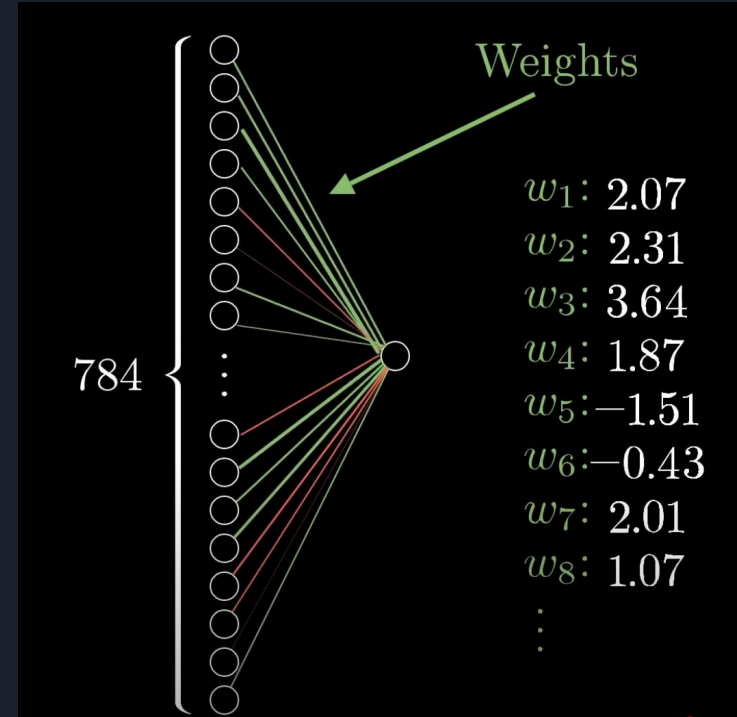
Project Design Theory: Network Structure

- Examples of hand drawn numbers
- Numbers are represented digitally by 28x28 drawings with grayscale values
- $28 \times 28 = 784$ makes up first layer of network
- Two hidden layers of 16 neurons each
- Output classification layer with 10 outputs



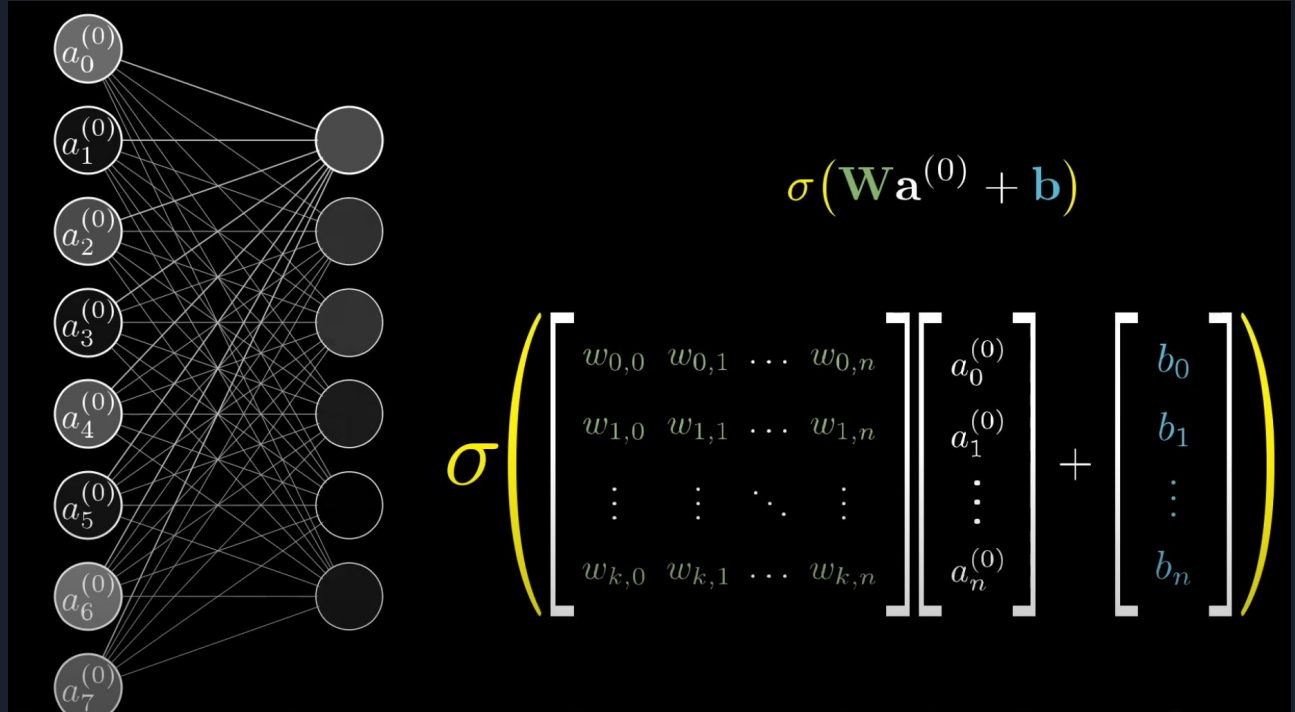
Project Design Theory: Neuron Weights

- Every neuron in each subsequent layer, is connected to every neuron in the previous layer
- These layer connections have weights
- In this example, the first hidden layer has 16 neurons, each having 784 weights



Project Design Theory: Network Parameters

- Weights
 - $784 \times 16 + 16 \times 16 + 16 \times 10 = 12,960$
- Biases
 - $16 + 16 + 10 = 42$
- Total number of parameters = 13,002
- Bias is how high weighted sum needs to be before the neuron becomes meaningfully active
- Sigmoid activation to squeeze numbers from 0 to 1



Project Design Theory: Cost Function

- Initial weights and biases are all completely random
- Throw initial training example through model
- Add up the squares of the differences between each of the models output activations and the desired value
- Consider average cost over all training examples

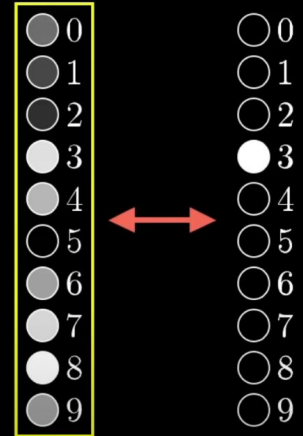
Cost of

3

3.37

$$\left\{ \begin{array}{l} 0.1863 \leftarrow (0.43 - 0.00)^2 + \\ 0.0809 \leftarrow (0.28 - 0.00)^2 + \\ 0.0357 \leftarrow (0.19 - 0.00)^2 + \\ 0.0138 \leftarrow (0.88 - 1.00)^2 + \\ 0.5242 \leftarrow (0.72 - 0.00)^2 + \\ 0.0001 \leftarrow (0.01 - 0.00)^2 + \\ 0.4079 \leftarrow (0.64 - 0.00)^2 + \\ 0.7388 \leftarrow (0.86 - 0.00)^2 + \\ 0.9817 \leftarrow (0.99 - 0.00)^2 + \\ 0.3998 \leftarrow (0.63 - 0.00)^2 \end{array} \right.$$

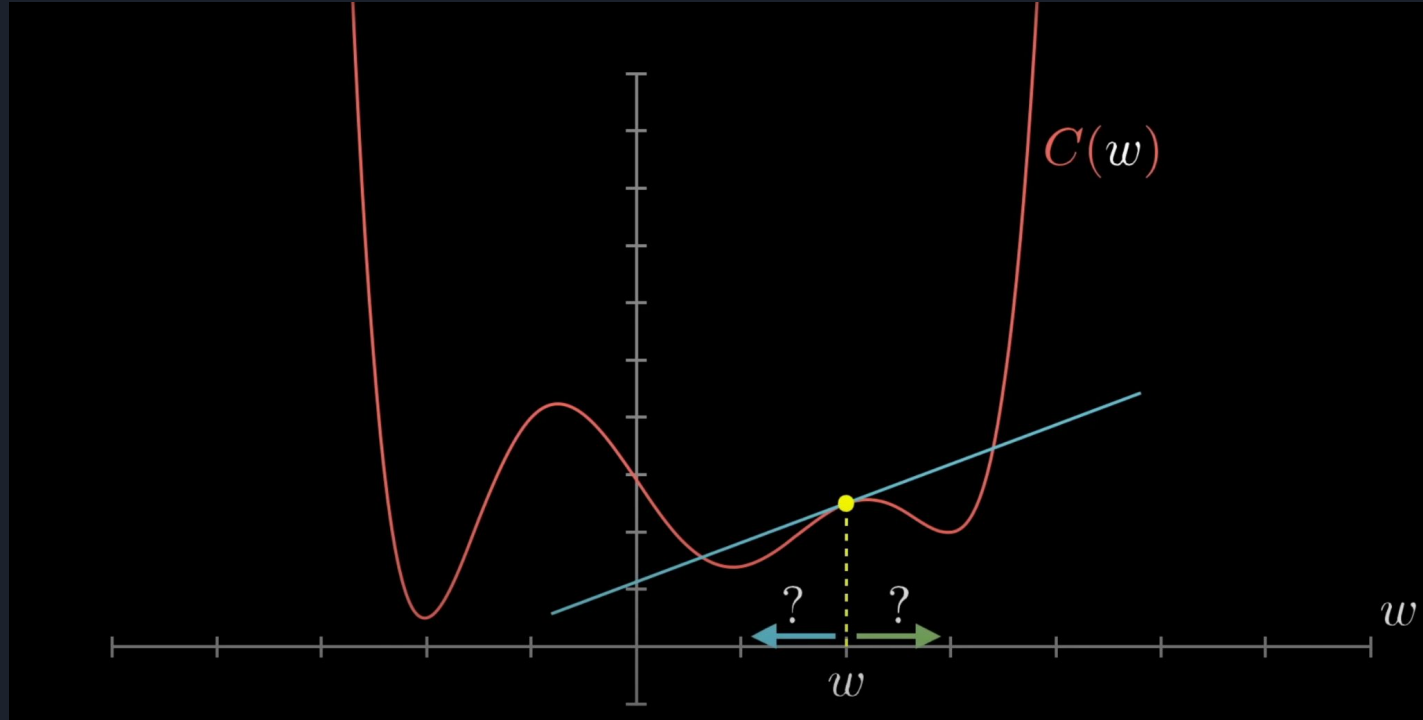
What's the “cost” of this difference?



Utter trash

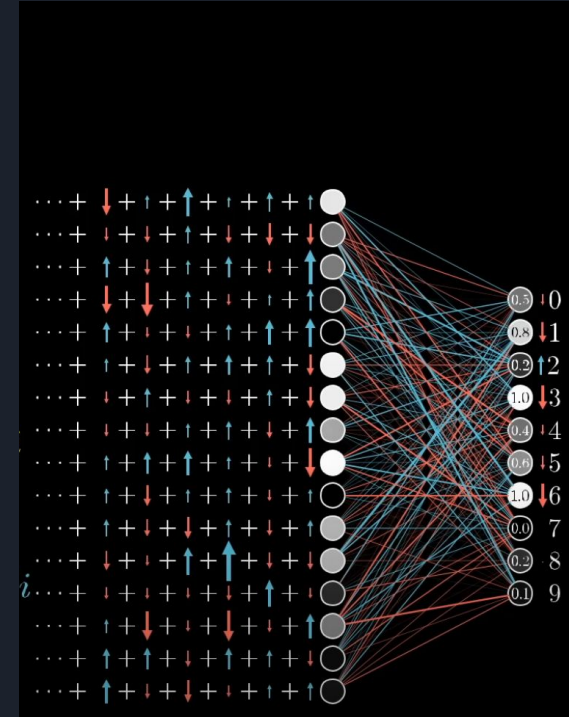
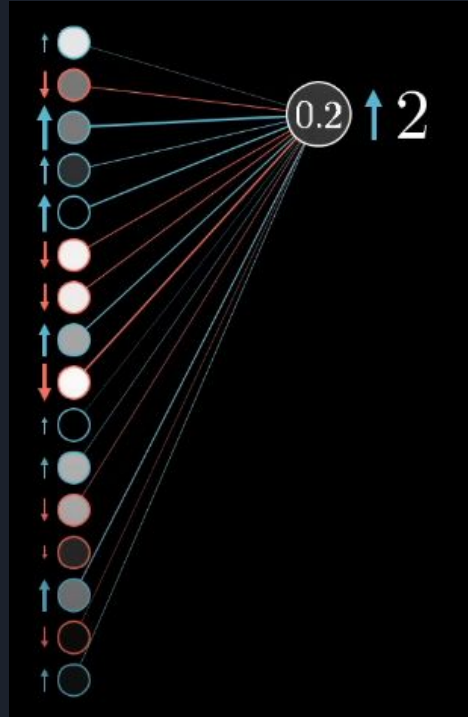
Project Design Theory: Gradient Descent

- Start at any random input
- Calculate the gradient to figure out which direction to make the cost function lower
- Take a tiny step in that direction
- Repeat the process until local minimum is found
- Finding a global minimum is too complicated



Project Design Theory: Back propagation

- Start with a training sample
- Find its correct class
 - Probability must increase
 - Look at all of the weights connected to that class
 - Positive weights must increase (blue)
 - Negative weights must increase (red)
- Find all of the incorrect classes
 - Probabilities for these need to decrease
 - Positive weights decrease (blue)
 - Negative weights increase (red)
- Add up desires of all output neurons for that layer
- Nudge weights by the sum
- Start process over at next layer
- Do that for every training example



Project Design Implementation

- Iris Dataset: Tabular Data
- Model Creation
- Custom Dataset Class
- Training Loop
- Model Ensemble



Project Design Implementation: Iris Dataset

- Tabular Data
- Similar Dataset to Signal Parameters
- Lacking Categorical Variables
- Very Small (only 150 examples)

	A	B	C	D	E
1	sepal.length	sepal.width	petal.length	petal.width	variety
2	5.1	3.5	1.4	0.2	Setosa
3	4.9	3	1.4	0.2	Setosa
4	4.7	3.2	1.3	0.2	Setosa
5	4.6	3.1	1.5	0.2	Setosa
6	5	3.6	1.4	0.2	Setosa
7	5.4	3.9	1.7	0.4	Setosa
8	4.6	3.4	1.4	0.3	Setosa
9	5	3.4	1.5	0.2	Setosa
10	4.4	2.9	1.4	0.2	Setosa
11	4.9	3.1	1.5	0.1	Setosa
12	5.4	3.7	1.5	0.2	Setosa
13	4.8	3.4	1.6	0.2	Setosa
14	4.8	3	1.4	0.1	Setosa
15	4.3	3	1.1	0.1	Setosa
16	5.8	4	1.2	0.2	Setosa
17	5.7	4.4	1.5	0.4	Setosa
18	5.4	3.9	1.3	0.4	Setosa
19	5.1	3.5	1.4	0.3	Setosa
20	5.7	3.8	1.7	0.3	Setosa
21	5.1	3.8	1.5	0.3	Setosa
22	5.4	3.4	1.7	0.2	Setosa
23	5.1	3.7	1.5	0.4	Setosa
24	4.6	3.6	1	0.2	Setosa
25	5.1	3.3	1.7	0.5	Setosa

Project Design Implementation: Model Creation

```
8 # Create a Model Class that inherits nn.Module
9 class Model(nn.Module):
10
11     # input layer (4 features of the flower) --> 4
12     # hidden layer 1 (number of neurons) --> 16
13     # hidden layer 2 (number of neurons) --> 16
14     # output (3 classes of iris flowers) 3
15
16     def __init__(self, in_features=4, h1=16, h2=16, out_features=3): # initializes parameters of network
17
18         super().__init__() # delegates init call to parent class nn.Module
19
20         self.fc1 = nn.Linear(in_features, h1) # creates fully connected linear layer
21         self.fc2 = nn.Linear(h1, h2) # creates fully connected linear layer
22         self.out = nn.Linear(h2, out_features) # creates output layer
23
24     def forward(self, x): # defines computations performed and flow of data
25         x = F.relu(self.fc1(x)) # relu activation on 1st layer
26         x = F.relu(self.fc2(x)) # relu activation on 2nd layer
27         x = F.softmax(self.out(x)) # softmax on final layer to get probabilities
28         return x
29
```

Project Design Implementation: Custom Dataset


```
8 # Creates a custom dataset class that reads in a dataframe
9 class IrisDataset(Dataset):
10     def __init__(self, dataframe):
11         dataframe = self.label_change(dataframe) # encodes categorical words to numbers
12         self.features = dataframe.drop(['variety', 'variety_encoded'], axis=1) # drops label data and saves numerical features
13         self.labels = dataframe['variety_encoded'] # assigns labels to encodings
14         self.tensor_transform() # changes numerical data to tensors
15
16     def __len__(self):
17         return len(self.labels) # returns length of labels
18
19     def __getitem__(self, idx):
20         features = self.features[idx] # defines how data is pulled from dataset
21         labels = self.labels[idx]
22         return features, labels
23
24     def label_change(self, dataframe): # changes labels to encodings
25         label_encoder = LabelEncoder()
26         dataframe['variety_encoded'] = label_encoder.fit_transform(dataframe['variety'])
27         return dataframe
28
29     def tensor_transform(self): # changes data to tensors
30         self.features = torch.FloatTensor(self.features.values)
31         self.labels = torch.LongTensor(self.labels.values)
32
33     def create_dataloader(dataframe, batch_size):
34         dataset = IrisDataset(dataframe)
35         dataloader = DataLoader(dataset, batch_size=batch_size)
36         return dataloader
37
38
39     def download_data(url):
40         my_df = pd.read_csv(url)
41         return my_df
42
43     def split_data(dataframe):
44         train_validate, test = train_test_split(dataframe, test_size=0.2, random_state=41)
45         train, validate = train_test_split(train_validate, test_size=0.2, random_state=41)
46         return train, validate, test
```


Project Design Implementation: Training Loop

```
4 def train_model(model, train_dataloader, validate_dataloader, criterion, optimizer, num_epochs):
5
6     # puts our model in training mode
7     model.train()
8
9     # create lists for training and validation loss per epoch
10    train_epoch_loss = []
11    validate_epoch_loss = []
12
13
14    # loop through the number of epochs you want to train
15    for epoch in range(num_epochs):
16
17        # creates list of batch losses
18        batch_losses = []
19
20        # iterate through dataloader
21        for data in train_dataloader:
22
23            # unpack data from dataloader
24            features, labels = data
25
26            # go forward and get a prediction
27            y_pred = model(features)
28
29            # measure the loss/error
30            loss = criterion(y_pred, labels) # predicted values vs the y_train
31
32            # neural network theory in practice
33            optimizer.zero_grad() # clears the gradients
34            loss.backward() # computes derivatives of loss function (gradient)
35            optimizer.step() # take step in direction of steepest descent of loss function
36
37            batch_losses.append(loss.item()) # append loss for data in batch
38
39        # keep Track of our losses
40        train_epoch_loss.append(sum(batch_losses)/len(batch_losses)) # calculates the average loss of all batches in an epoch
41        val_loss = validation_loop.validate_model(model, validate_dataloader, criterion) # calculates the validation loss of the model
42        validate_epoch_loss.append(val_loss.item()) # saves validation loss after each epoch
43
44        # print metrics
45        print(f'Epoch: {epoch} Training Loss: {train_epoch_loss[epoch]} Validation Loss: {validate_epoch_loss[epoch]}')
46
47    return model, train_epoch_loss, validate_epoch_loss
48
```


Project Design Implementation: Model Ensemble



```
36 class ModelEnsemble():
37
38     def __init__(self, models_list, test_dataloader): # initilize the model ensemble
39         self.models_list = models_list                # define the list of models to be used
40         self.test_dataloader = test_dataloader        # define the test_dataloader
41         self.run_inference()
42         self.majority_vote()
43
44     def run_inference(self):                            # get result from each model
45         outputs_list = []                             # create list for all model outputs
46
47         for model in self.models_list:                 # loop through list of models
48             _, _, _, labels, outputs = evaluate_model.test_model(model, self.test_dataloader) # get outputs from a model
49             outputs_list.append(outputs.flatten(start_dim=0, end_dim=1)) # go from 3d to 2d by flattening dimension 1
50
51         self.model_outputs = outputs_list # save model outputs
52         self.labels = labels.flatten()    # save labels
53
54
55     def majority_vote(self):
56         model_output_sum = sum(self.model_outputs)      # sum all of the model outputs
57         ensemble_output = torch.argmax(model_output_sum, dim=1) # find the label of the highest probability
58         self.ensemble_output = ensemble_output          # define that label as the output
59
```

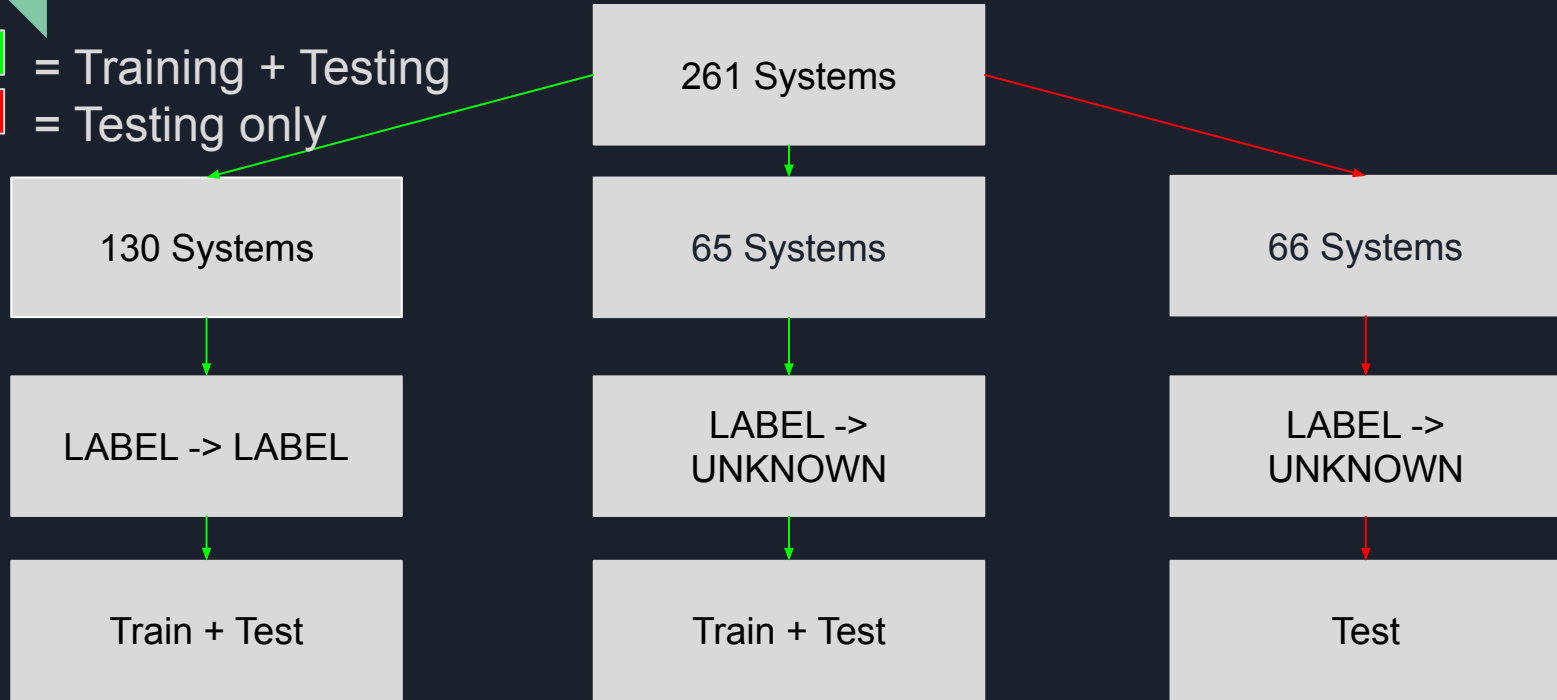


Neural Network For system Identification: Summary

- Queried Database for our Intercepts
 - Queried a minimum of 5,000 and a maximum of 10,000 intercepts
 - Signals had varying amounts of intercepts in the database
- Every intercept had an original system label
 - Total of 261 systems
 - Kept original system label for first half (130 systems)
 - Second half changed original system labels to 'UNKNOWN' (a total of 131 systems)
 - Used first half (65 unique systems) of the 'UNKNOWN' data to train the algorithm
 - Resulted in a category that had a wide variety of signals it has seen
 - The idea was to any signal the model hadn't been trained on would be dumped here
 - We used the second half (66 unique systems) of the 'UNKNOWN' data to test the algorithm
 - These were systems the model has never seen examples of
 - These were supposed to represent new systems in the electromagnetic environment
 - The hope was these would be dumped into the 'UNKNOWN' category

Diagram

 = Training + Testing
 = Testing only



System had 131 outputs: 130 unique system labels, then 1 for 'UNKNOWN'



Neural Network For system Identification: Results

- Final result metrics
 - Overall classification testing accuracy was 95%
 - On the training data that had the original system label (130 systems)
 - Most categories had 97-100% accuracy
 - Model performed very well on these
 - On the training portion of 'UNKNOWN' data
 - Model performed well 97%+ accuracy
 - On the training data the model had never seen
 - Supposed to be classified as unknown
 - Using the model ensemble, achieved roughly 76% accuracy
 - Means some of these systems were misclassified as one of the original system labels



Challenges

- Acquiring Development Software
 - Solved by investing in relationship with software team
 - Got approvals quickly and efficiently
 - Frequently checked in on tickets
- Locating Hardware to train algorithm
 - Searched multiple avenues for potential solutions
 - Poor: Acquire hardware at squadron
 - Best: Acquire account for free HPC hosted by Air Force Research Lab
 - Found a POC, reached out, got in touch with accounts team
- Dealing with Unknown Signals
 - Model was trained on subset of known systems
 - Needed a solution for systems the model was not trained on
 - Created unknown category for all systems the model wasn't trained on
 - Poor: Tricky problem, started with tweaking the probability threshold for classification
 - Robust: A better solution was to use a model ensemble, with a default of 'UNKNOWN'



Lessons Learned

- Take ownership and responsibility
 - Best way to grow technically
 - Increased motivation and performance
- Leverage other people's skills
 - Humility is a great tool, know what you don't know
 - Find smart people, ask them for help
- Follow up consistently with requests
 - People have their own agendas
 - Make sure they don't forget about yours
- Be vocal about what you're working on
- Jump into learning new things
 - Headfirst is the best way to go
 - Learned Python, Linux, SQL, MATLAB
- Briefing leadership
 - Do not bog them down on technicals
 - Keep briefings very short, 3 slides if possible
 - Clearly define WHY
 - Focus on impact, result, and metrics
 - Give them timelines and future plans

Takeaways: Could recreate project in 1/3rd of the time (3-4 months). In a more technically advanced environment, I'd lean more heavily on other professional developers



Final Impact of Project

As of my departure, project has not been deployed. Still several positive takeaways

- Development environment on high side network
 - Python, Git, VScode approvals
 - On government highside, approvals take weeks sometimes months
- Multiple resources were located
 - High Performance Compute Cluster
 - NSA python package repository
- Machine Learning Foundation
 - Github repository
 - Initial code and project created for further development
- Additional Engineers Setup for success
 - After I left, 2 engineers had HPC accounts, and development software installed
 - Those engineers are responsible for getting projected deployed



Why SpaceX and Starshield?

- Similar Mission
 - Satellites and data
 - Government focused
- Technical Expertise
 - Previous employment lacked technical experience
 - SpaceX has some of the best engineers in the world
 - Hoping to learn from peers
- Ambition, purpose, and drive
 - Previous employment lacked ambitious, driven, lifelong learners
 - Takes their missions seriously, there's a lot on the line, everyone's work matters
 - Culture of excellence and merit